Chapter Four: Contents

(Route Planner – LA-UR-00-1767)

Chapter Four: Figures

Chapter Four: Tables

Chapter Four—Route Planner

1. INTRODUCTION

1.1 Overview

As its name implies, the Route Planner module generates routes for travelers. Each traveler, including transit drivers, itinerant travelers, and truck drivers, receives an individual travel plan. Once the plans are generated for all travelers, they are simultaneously executed in the Traffic Microsimulator.

Constraining the routes between different locations are (1) the network, which represents the metropolitan region being studied, and (2) the preferences of individual travelers. Information about each traveler’s activities (contained in Chapter 3: (*Activity Generator*)) is used to create trip requests. A trip request consists of three parts:

- the origin and destination of the trip,
- the ranges for the preferred starting time and ending time, and
- the travel mode choice.

Given in the form of a string of characters, the travel mode choice defines the allowed modes of travel and their order. The Mode Preference File (configuration file key: `MODE_MAP_FILE`) defines the mapping between mode choices (as used in the activity file) and strings (as used in the Route Planner). Additional information about vehicles that a particular traveler may use is contained in the Vehicle File (configuration file key: `VEHICLE_FILE`). Fig. 1 shows the data flow the Route Planner uses to generate plans for individual travelers.

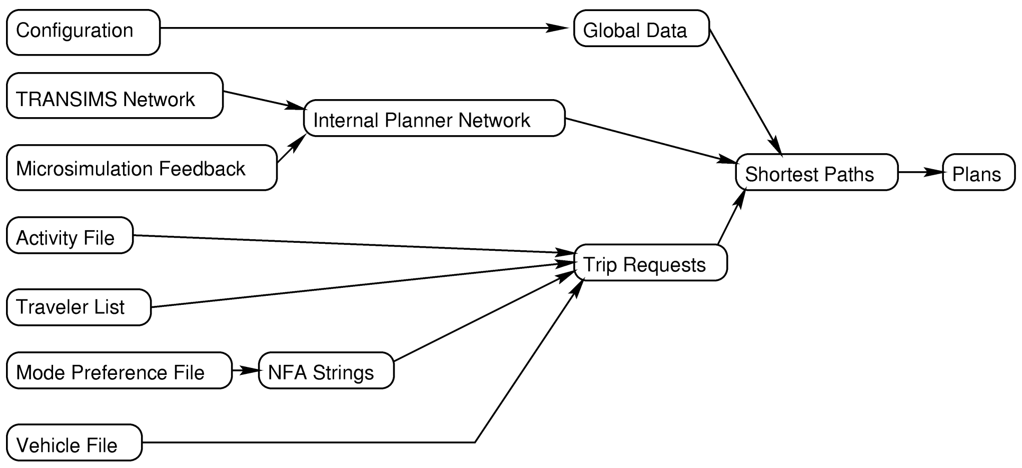

Fig. 1. Data flow diagram that shows how the TRANSIMS Route Planner generates travel plans for travelers.

1.2 TRANSIMS Network

The TRANSIMS Network provides information about streets, intersections, signals, parking, activity locations, and transit in a road transportation network. This information is used to construct the internal planner network. The internal network is time dependent—that is, travel on a link may incur different delays at different times of the day. The information about delays on links derived from the Traffic Microsimulator output and provided in the Feedback File (configuration file key: `ROUTER_LINK_DELAY_FILE`), which specifies the mean delays on each link over 15-minute intervals. If the delay for a particular interval is not given, then the free speed delay is used.

The Route Planner’s core is Dijkstra’s shortest-path algorithm, with extensions for time-dependent delays and travel mode choice constrained paths.

The internal network and trip requests are given to the path-finding routine, which creates routes and outputs them in the form of plans. To ensure an efficient implementation of the Route Planner, we use threads that enable the parallel execution of several copies of the path-finding algorithm. Each thread uses the same copy of the network to create plans from different trip requests.

1.2.1 Route Planner Major Input/Output

Fig. 2 shows the Route Planner’s major inputs and outputs. The major inputs to the Route Planner are

- Transit routes and schedules,
- the activities list,
- the TRANSIMS multimodal network,
- the vehicle file, and
- link travel times.

The major outputs of the Route Planner are

- the plan list, and
- the anomalous activity list.

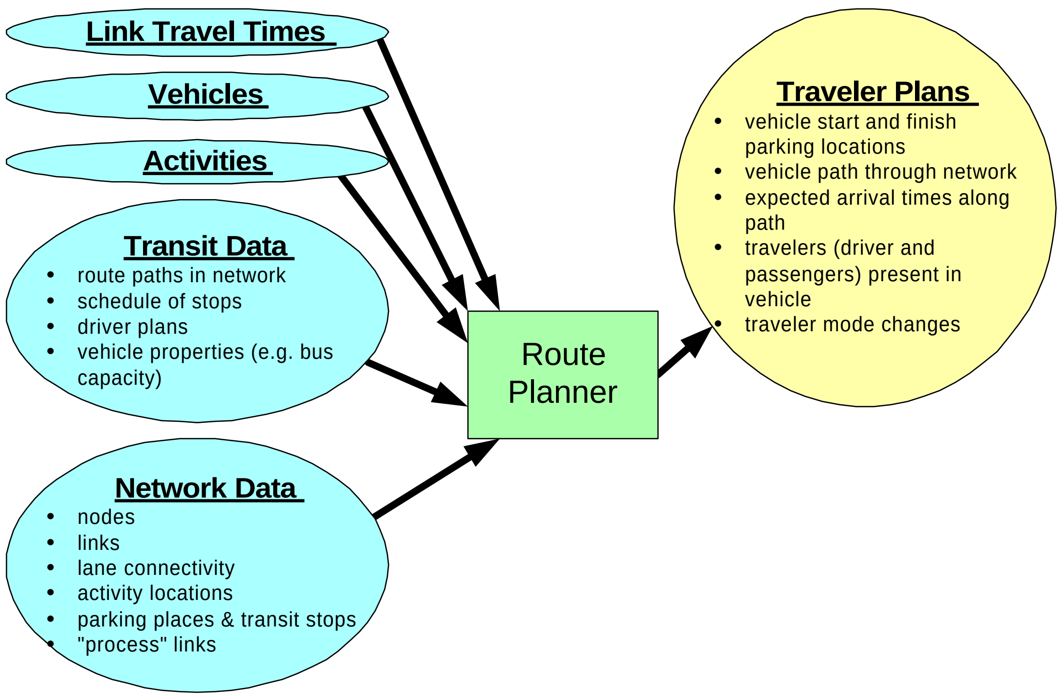
Link Travel Times
Vehicles
Activities
Transit Data
route paths in network
schedule of stops
driver plans
vehicle properties (e.g. bus capacity)
Network Data
nodes
links
lane connectivity
activity locations
parking places & transit stops
"process" links
Route Planner
Traveler Plans
vehicle start and finish parking locations
vehicle path through network
expected arrival times along path
travelers (driver and passengers) present in vehicle
traveler mode changes

Fig. 2. The major input to the Route Planner includes the following data: (1) TRANSIMS Network, (2) activities, (3) transit, and (4) vehicle information from the synthetic population data.

2. Route Planner Description

2.1 Overview

The Route Planner computes the “shortest” path for each traveler in the system (this is subject to mode constraints). Each link within the network system has a cost associated with it. Accordingly, the shortest path can be interpreted as least cost, for some generalized meaning of cost. Constraints are provided by criteria such as mode preferences for different legs of the trip.

Costs for a link can be computed simply with input, such as an estimated time delay.

There are also more sophisticated ways to calculate costs. For example, they can be calculated based on several variables, including time delays and the actual monetary costs of a link. More abstract variables can be used, such as a penalty for traveling through construction areas, or traveler demographics, such as household income level.

2.2 Distinguishing Features

The Route Planner has three distinguishing features.

2.2.1 Individual Plans

Plans are computed for each individual traveler in the population, based on that individual’s activity demands and preferences. Such computations enable each traveler to have an individualized view of the transportation system. Accordingly, costs associated with links in the network are computed separately for each traveler.

2.2.2 Per Link Time-Dependant Delay

Link costs are computed in a time-dependent manner that can account for time delays resulting from actual travel conditions, such as peak-hour congestion.

2.2.3 Travel Mode Constraints

The Route Planner abides by any mode preferences contained in the activity files. Thus, if the activity files specify that a traveler will (1) walk, (2) take a car, and (3) walk again between two desired activities, the Route Planner will produce a plan (if feasible) that ensures these modes are used in this sequence.

2.3 Terminology

2.3.1 Traveler Plan

A *traveler plan* consists of a set of trips that carries the traveler through his or her desired activities. A *trip* consists of a set of contiguous legs. Activities of a given duration may separate trips. A *leg* consists of contiguous nodes and links that are traversed with a single travel mode. For example, a trip may consist of three legs:

- walking,
- transit, and
- walking.

A traveler plan would consist of a trip (1) from home to work, (2) work to shopping, and (3) a trip from shopping to home.

2.3.2 Transit Vehicle

From the point of view of the Route Planner, a *transit vehicle* is considered to be any vehicle that makes scheduled stops along a predetermined route. Buses, trains, and streetcars are all considered transit vehicles, whereas a taxi would not be considered a transit vehicle.

2.3.3 Trip Request

A request for travel to be planned by the Route Planner, *trip request* consists of a starting location, a destination location, a starting time, and a mode string.

2.3.4 Mode String

A mode string contains a list of travel modes that must be used in the order given along the path from source to destination.

2.4 Travel Modes

There are twelve individual modes available within TRANSIMS. The modes and their corresponding mode letter are show in Table 1. Bike mode is routed at a faster speed on the walk network. Transit mode allows travel on any type of mass transit system (bus, rail, streetcar, or trolley). Transit/Walk allows walking in between transit routes, allowing transfers between different types of transit that may not use the same transit stop. The transit/walk mode is currently not implemented.

Magic mode is an unrouted mode, for magic moves a walk plan whose start and stop times are taken from the times given in the activities. Its intended use is to support travel modes that are not supported by the Route Planner and/or the microsimulation, such as school busses. The mode string wcwxw, where x is one of b, l, g, p, y, s, t, or T, is used for park-and-ride.

Any mode string consisting of the mode letters in Table 1 can be planned. Some are meaningless because they will never produce paths (e.g., cb—because a traveler must walk from a parking location to a bus stop). Some others may not produce exactly what is expected: b restricts the plan to boarding only one bus because the process links between bus line nodes and bus stops are traversed in walk mode.

Table 1. Currently recognized travel mode letters.

Mode	Mode Letter
Walk	w
Bike	i
Car	c
Bus	b
Light Rail	l
Regional Rail	g
Rapid Rail	p
Trolley	y
Street Car	s
Transit	t
Transit/Walk	T
Magic Mode	k

2.5 Trip Requests

2.5.1 Generating Trip Requests from Activities

The activity, vehicle, and mode files are used to generate trip requests, which are then planned. In the activity file (configuration file key: `ACTIVITY_FILE`), travelers' mode preferences are given by integers. Their meaning is defined by the mode file, which gives the correspondence between these integers and mode strings used by the Route Planner.

The Route Planner uses the household file (configuration file key: `ROUTER_HOUSEHOLD_FILE`) to determine the travelers for which plans should be generated. If this value defines a file that can be opened, then only the travelers belonging to households whose IDs are listed in this file will be routed. Otherwise, the Route Planner routes all of the travelers in the activity file. All travelers in a single household are planned together because they may share transportation or activities. The Selector/Iteration Database uses this file as part of the feedback mechanism that enables a portion of the population to be re-planned.

Plans are generated in response to trip requests for a traveler. Trip requests come from the activity file. For every traveler, each pair of consecutive activities at different locations generates a trip request. A trip request consists of a source activity location; a destination activity location; constraints on the start time, end time, and duration; and the travel modes that are allowed. A trip request is satisfied by a plan, in the form of a trip made up of unimodal legs. Travel plans are separated by activity plans.

The activities of each traveler are split into legs that define either activities (activity legs), or travel (transportation legs). Activity legs begin and end at the same activity location. Transportation legs begin and end at different activity locations. The activity legs are not planned, and are written into the plan file using the times from the activity file. Travel plans are created for the transportation legs. If a transportation leg is multimodal, it is further split up into unimodal sections, which are planned as separate legs of a trip.

If a planned trip uses a car, the vehicle file (1) is examined to find the location of the car, and (2) the trip is split. The first mode string ends with the last symbol before `c`, and the destination of the first part of the trip is the parking location where the car is located.

The second part of the trip starts there (with the mode `c`) and ends at the original trip's destination. The two parts are planned separately then written out consecutively in the plan file.

2.6 Parking

Because TRANSIMS tracks the movements of each individual throughout the simulation, the Route Planner retains the location of each household's vehicles. This enables an individual from a household to drive to a parking location, walk from the parking lot to work, then return to the same parking location to retrieve the vehicle for the trip home.

Currently the Route Planner will pick a parking location adjacent to the destination activity location for the trip as the destination parking location. If there is no adjacent parking location, the Route Planner will display a warning and skip the remainder of the traveler's activities. In this case, adjacent means that there is a process link from the ending parking location to the ending activity location. This restriction will be removed in a future version of the Route Planner.

2.7 Shared Rides

A shared ride is one in which a passenger travels in an automobile driven by another traveler. Currently only shared rides in which the passenger and the driver are part of the same household are supported. The driver trip request is planned as usual. Any passenger trip requests are fulfilled by using information from the driver plans.

The driver and passenger trip requests are matched according to the following procedure. The trip requests for a passenger with a particular driver are listed in the order that they occur in the activity file. The driver trip requests that include the passenger are also listed in activity file order. The driver and passenger trip requests are matched in order according to these lists. This process is repeated for every combination of driver and passenger that occurs in a household. If there are not enough driver trip requests to satisfy all of the passenger trip requests, the passenger activity is listed in the anomalous activity file with an anomaly type of *Invalid Shared Ride*. The condition where there are too many driver trip requests is not detected.

Because of interdependencies between travelers (a driver in the morning may be a passenger in the afternoon), a passenger activity may be planned before the corresponding driver activity. Room for the passenger trip is left in the plan sequence according to the desired activity times. If the driver trip is longer than expected (e.g., because of congestion), there may not be enough time between activities in the passenger plan. In this case, the activity leg following the passenger trip is shortened to accommodate the transportation leg and the activity is recorded in the anomalous activity file with the anomaly type *Invalid Shared Ride Time*. If the passenger trip extends past the end of the following activity, the remaining activities for the passenger are not planned.

2.8 Anomalous Activity File

There are currently four types of anomalous activities recognized by the Route Planner*:* *No Path*, *Invalid Time*, *Invalid Shared Ride*, and *Invalid Shared Ride Time*. For each activity in which an anomaly is detected, a line is rewritten to the anomaly activity file (configuration file key: `ROUTER_PROBLEM_FILE`). These data can be used by the Selector/Iteration Database module to request new activity characteristics for the traveler on the household.

The first seven fields (see Table 2) of each line are the same for each type of anomaly. These fields describe the activity for which an anomaly was detected, the trip generated for this activity, the type of anomaly detected, and the number of anomaly-specific fields remaining. If no trip was generated for this activity, then the TripId and LegId fields are set to -1.

Table 2. Anomalous activity file common fields.

Field	Description
HouseholdId	ID of the anomalous household
TravelerId	ID of the anomalous traveler
ActivityId	ID of the anomalous activity
TripId	ID of the trip generated by this activity
LegId	ID of the first leg generated by this activity
ProblemType	Type of anomaly: 1-*No Path*, 2-*Invalid Time*, 3 - *Invalid Shared Ride*, 4 – *Invalid Shared Ride Time*
Number of data fields	Number of remain fields, varies by anomaly type

A *No Path* anomaly takes place when a trip request cannot be satisfied because a path from the source location to the destination location could not be found which obeys the time and mode constraints. Common reasons for this anomaly include no connectivity between the source location and the destination location, and no transit vehicles running after the start time. The *No Path* anomaly includes information about the source and destination accessories, the mode, and the start time of the transportation leg. When a *No Path* anomaly is detected, no plan is generated, and the rest of the activities for this traveler are skipped. Table 3 provides the *No Path* fields.

Table 3. No Path fields.

Field	Description
SourceLocation	Source Location of the anomaly trip
SourceType	Source Location type of the anomaly trip
DestinationLocation	Destination Location of the anomaly trip
DestinationType	Destination Location type of the anomaly trip
Mode	Travel mode of the anomaly trip
StartTime	Time the anomaly trip should start

An *Invalid Time* anomaly occurs when the actual time used by the Route Planner does not fit within the bounds specified by the activity. The start time, end time, and duration are checked for consistency with the ranges given in the activity. A separate line in the anomalous activity file is output for each one of these times that is inconsistent. The line contains the type of the inconsistency, the lower and upper bound from the activity file, and the actual value used by the Route Planner. A plan is generated for the anomalous activity using the inconsistent times. Table 4 provides the *Invalid Time* fields.

Table 4. Invalid Time fields.

Field	Description
TimeType	Field which has the anomaly 0-Start, 1-End, 2-Duration
LowerBound	Distribution lower bound
UpperBound	Distribution upper bound
Actual	Actual value used

An *Invalid Shared Ride* anomaly occurs when the driver activities and passenger activities do not match up. Currently, only the condition where there are too few driver activities for the number of passenger activities is detected. When this anomaly is detected, no plan is generated for the passenger and the rest of the passenger's activities are not planned. The driver activities are planned as usual. No extra fields are output for this anomaly.

An *Invalid Shared Ride Time* anomaly takes place when the transportation leg for a passenger shared ride takes longer than the time between the two surrounding activity legs. If the trip extends past the upper bound of the following activity's start time, but not past the following activity's end time, an *Invalid Shared Ride Time* entry is created in the anomalous activity file and the rest of the passengers trip requests are planned. If the trip extends past the end time of the following activity, an *Invalid Shared Ride Time* entry is created in the anomalous activity file, and no further trips are planned for this traveler. The *Invalid Shared Ride Time* anomaly contains the arrival time of the passenger shared ride trip, the upper bound of the start time of the following activity, and the end time of the following activity. Table 5 provides the *Invalid Shared Ride Time* fields.

Table 5. Invalid Shared Ride Time fields.

Field	Description
Arrival Time	Arrival time of the passenger shared ride trip
Start Time Bound	Upper bound of the starting time of the activity leg following the passenger shared ride trip
Stop Time	The stop time of the activity leg following the passenger shared ride trip

2.9 Network Layers

The Route Planner conceptually views the network as a set of interconnected, unimodal layers (see Fig. 3). In other words, a separate layer exists for each mode letter in the mode string. At certain designated locations (which becomes nodes in the Route Planner's view of the network) in each layer, a special link, called a process link, connects one or more of the unimodal layers to another. These process links allow intermodal transitions to take place.

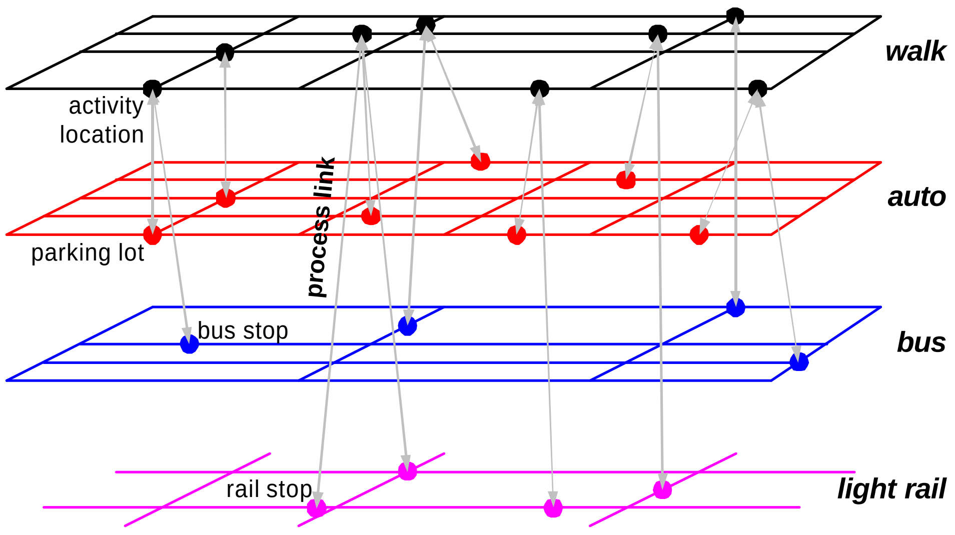

Fig. 3. A high-level depiction of the various layers used by the Route Planner. From individual traveler preferences and constraints contained in the synthetic population and activities data blocks, the Route Planner plans for trips that consist of multiple modal legs (e.g., walk-car-walk). Constructing multiple layers in which each layer can be encoded as a different uni-modal network allows for the efficient calculation of trips constrained by modal sequences. Also shown are the process links connecting the uni-modal networks.

The process links are considered to be part of the walking layer. The layers are constructed from the TRANSIMS Network. Delays for each link in each layer are computed by a link delay function, which is time-dependant.

Conceptually, layers are associated with modes of travel. In this view, there are three types of layers in the network:

1) A street layer, which consists of all links between intersections, with added parking locations.

2) A walk layer, which consists of all streets that can be walked along. It contains activity locations. However, the parking locations and transit stops that belong to the other two layers are only accessible from activity locations.

3) Transit stops and links to transit layers, which can be traversed in transit (bus or light rail) modes only. There is a separate layer for each type of transit vehicle (e.g., bus and light rail), and a layer for each transit route via process links.

In Fig. 4, nodes A and B are street nodes. They correspond to original TRANSIMS Network nodes. Nodes P and Q are parking locations, whereas R and S are activity locations. Nodes C and D are transit stops. The links between layers are called process links.

Conceptually, nodes A and B appear in two different layers, even though these appearances correspond to the same TRANSIMS nodes. The reason for this is that even though we might be in the same geographic location (whether in a street or walk network), we cannot change from the street to the walk network without visiting an activity location and using a process link.

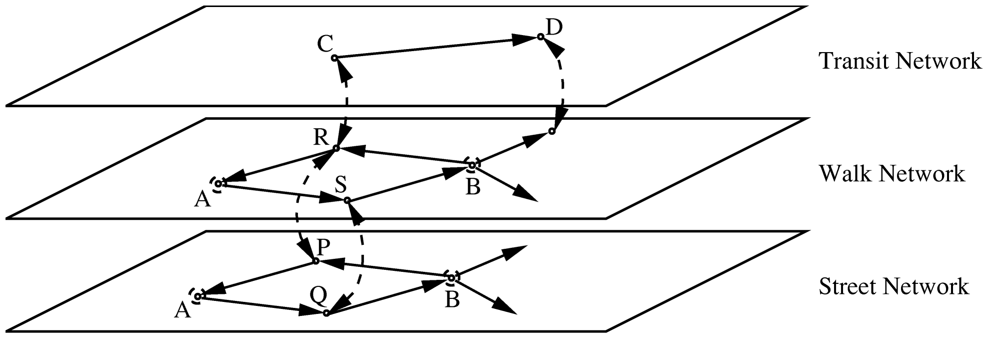

Fig. 4. Conceptual diagram of the planner network, in which parking accessories (P,Q) are in the street layer, activity locations (R,S) in the walk layer, and transit stops (C,D) in the transit layers.

2.9.1 Example

The Activity Generator provides mode preferences for each trip. This information is captured in simple, alphabetical expressions. For example, wcw represents a trip that breaks down as follows:

- w = a walking leg from a traveler’s house to his or her car.
- c = a car leg to parking at the place of work.
- w = a walking leg from the parking lot to his/her actual work location.

For the first leg of the trip (the walking leg), the Route Planner searches for possible paths within the walking layer of the network to obtain a walking route from the home to the parking location of the individual's vehicle. After the walking path is found, a series of least-cost driving links is found to obtain a route to a parking location near the work location. A walk route is then developed to move the traveler from the parking lot to the work activity location.

The last two legs of the above route highlight the Route Planner's capabilities. Once the search algorithm is in the car layer, it chooses additional links from the car layer or parks the vehicle and chooses links from the walking layer—whichever is lower in cost. The Route Planner ensures that the final link is a walking link in this example.

Trips that cannot be feasibly planned or that contain questionable legs are marked and provided as output from the Route Planner in the form of the Route Planner anomalous activity file (configuration file key: `ROUTER_PROBLEM_FILE`). These are fed back to the Activity Generator to choose a new activity time or location or mode of travel.

3. ALGORITHM

3.1 High-Level Description

To maintain computational efficiency, the TRANSIMS Network is converted to an internal route network (this is described in Section 3.2 of this chapter). The internal network represents a weighted, directed graph. The graph's nodes represent intersections and accessory locations (such as parking accessories, activity locations, and transit stops); the arcs (directed edges) represent travel possibilities between node pairs. Internally, all links are uni-directional. Bi-directional TRANSIMS links are represented by two separate links in the Route Planner.

The algorithm underlying the TRANSIMS Route Planner is the classical Dijkstra's algorithm, which finds the shortest paths in a weighted, directed graph. This algorithm can be viewed as a breadth-first search of the graph, starting at the origin node and visiting the other nodes in the order of their (shortest-path) distance from the origin. The actual algorithm used is a direct generalization of Dijkstra's algorithm. In fact, it can be viewed as Dijkstra's algorithm on a larger graph. In full generality, it is described by Barrett, Jacob, and Marathe.[1]

To increase the Route Planner's speed, the work can be distributed among several computer processors (when they are available). We do this by using POSIX threads (pthreads). The Route Planner uses one master thread and one or more worker threads.

The master thread creates (1) a queue of household trip requests, and (2) a queue of plans generated from the household trip requests. Each request contains the trip requests for each member of a particular household. The entire household is grouped together to enable shared rides. The master thread also constructs the internal planner network, which is accessed by each worker thread; it also is responsible for writing the plans placed in the plan queue to the plan file.

Each worker thread removes a household trip request from the trip request queue and generates plans for the requests. The generated plans then are placed in the plan queue for output by the master thread.

3.2 Internal Network

The Route Planner uses information from the TRANSIMS Network and some other files to create the Route Planner Internal Network representation, hereafter referred to as the internal network. The reason for the internal networks is to increase the efficiency of the path finding algorithm.

[1] C. Barrett, R. Jacob, and M. Marathe: "Models and Efficient Algorithms for Routing Problems in Time-dependent and Labeled Networks," Proc. 6th Scandinavian Workshop on Algorithm Theory, LNCS 1432.

3.3 Terminology

3.3.1 Node

Node: A physical location in the TRANSIMS Network, such as an intersection, activity location, or bus stop.

3.3.2 Link

Link: A street connection from the TRANSIMS Network. Every link has a delay, a layer, and one or more modes of travel associated with it.

3.3.3 Edge

Edge: A connection between two nodes. Each edge has an associated link and a fraction of the link that it represents.

One of the main differences between the TRANSIMS Network and the internal network is that the edges in the internal network are all uni-directional. Any bi-directional links in the TRANSIMS Network are converted to a pair of uni-directional links in the internal network, one in each direction.

There is a node in the internal network for each node in the TRANSIMS Network, as well as each parking location, activity location, and transit stop.

Each link in the TRANSIMS Network can have accessories attached to it. These accessories represent activity locations, parking, and transit stops, and become additional nodes in the internal network. Transit stops are described in more detail below. Activity locations are placed on the layer specified in the TRANSIMS Network activity location table, while parking locations are always placed on the street layer. For ease of discussion, the following examples assume that all activity locations are placed on the walk layer.

The TRANSIMS Network representation of two nodes with a connecting bi-directional link is shown in Fig. 5. There are six parking locations and five activity locations, connected by process links as shown. One of the parking locations has been designated as a commuter park-and-ride lot.

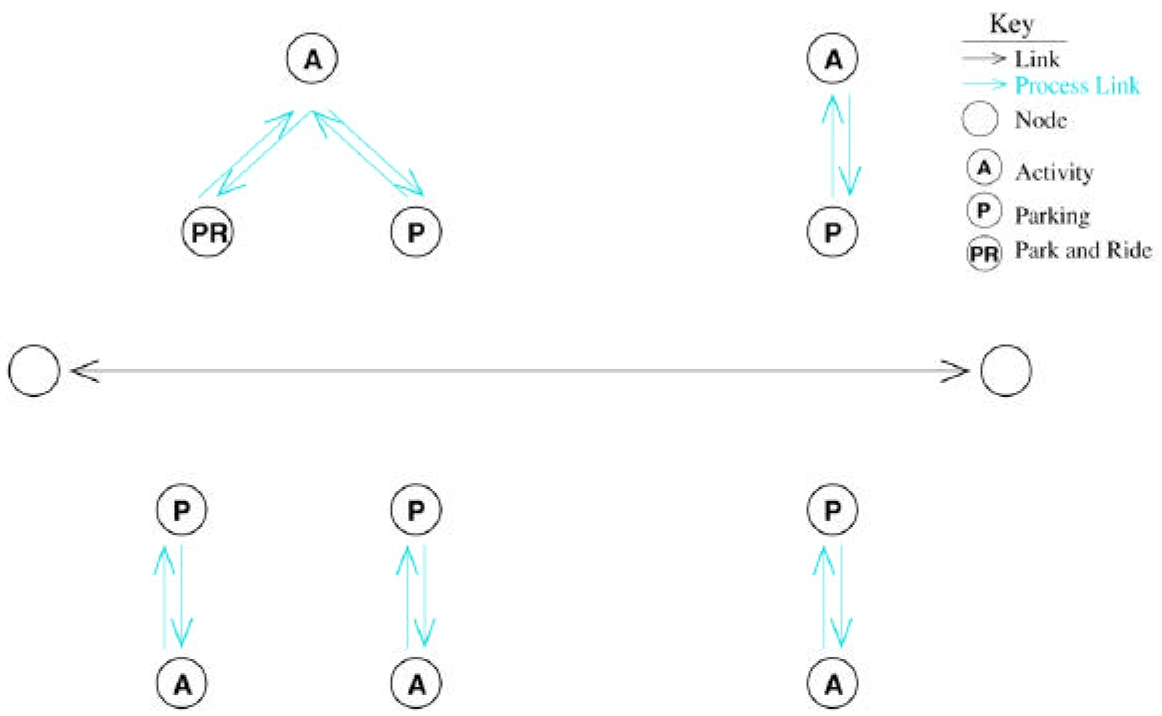

Fig. 5. The TRANSIMS Network representation of two intersection nodes with a connecting bi-directional link.

The corresponding nodes and edges of the internal network representation are shown in Fig. 6. The single link between the two intersection nodes in the TRANSIMS Network has been transformed into four uni-directional links. There is one link in each direction in the street network, as well as a link in each direction in the walk network. If a traveler parks at a park-and-ride lot, the Route Planner ensures that the traveler passes through the park-and-ride layer when going from the parking locations to the activity location.

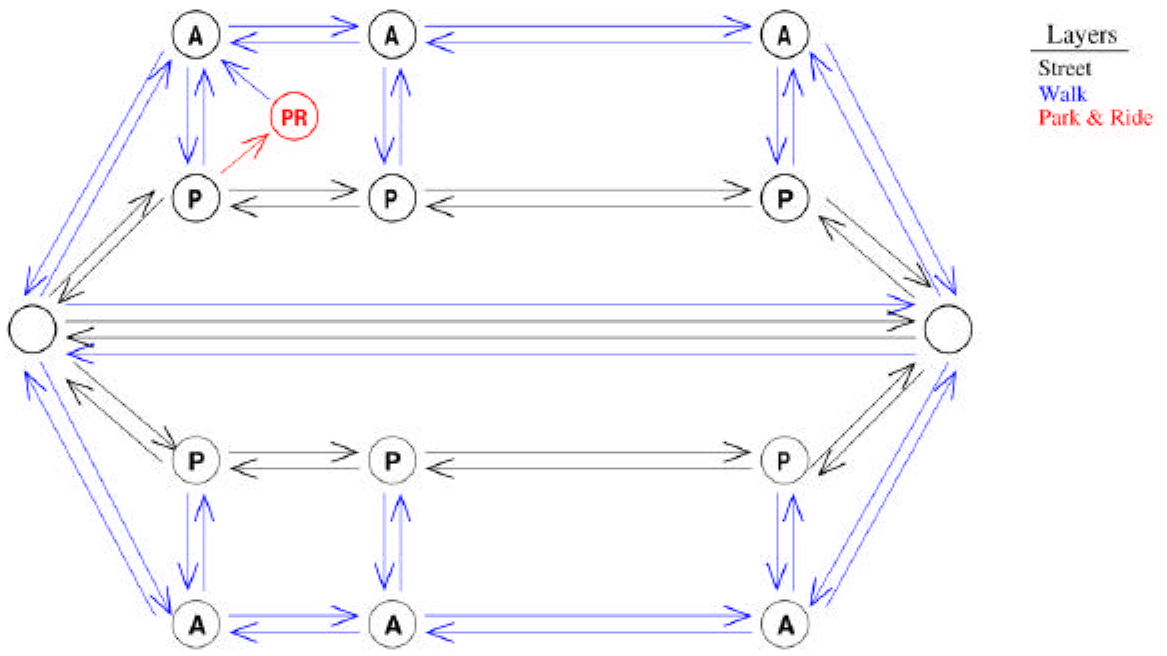

Fig. 6. The corresponding nodes and edges of the internal network representation.

The edges connecting the two intersection nodes have fraction 1.0. The edges that connect the parking accessories are assigned fractions according to the length of the link and the offset of the parking location from the node. The edges connecting the activity locations are similar.

If a link in the TRANSIMS Network does not allow walking, such as a freeway link, any activity locations in the walk layer are still connected by edges. However, no edges are placed between the activity locations and the intersection nodes.

Fig. 7 shows a TRANSIMS Network that is similar to the one shown in Fig. 5, with the exception of a uni-directional link in place of the bi-directional link.

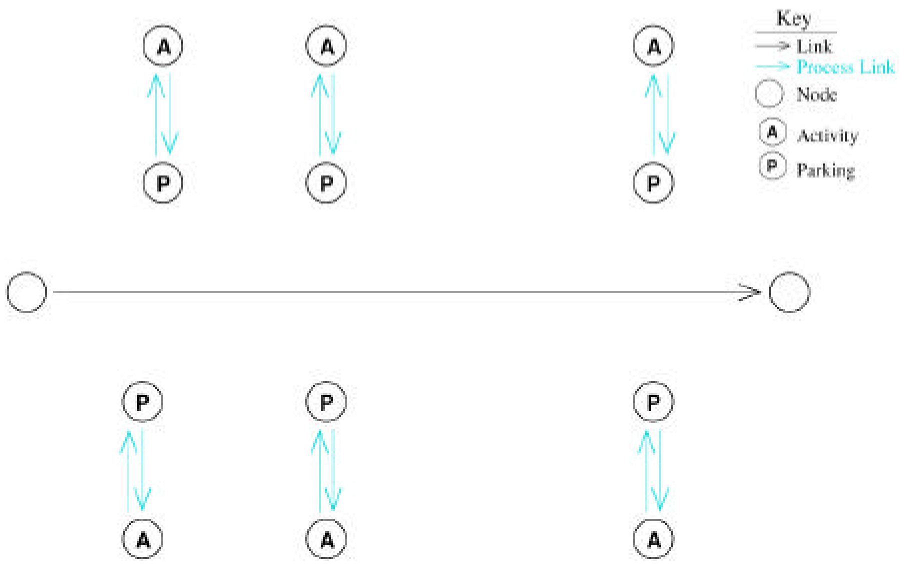

Fig. 7. This TRANSIMS Network is similar to the one shown in Fig. 5, with the exception that this one has uni-directional link in place of the bi-directional link.

As can be seen from Fig. 8, there are only edges in one direction on the street layer. However, there are still edges in both directions on the walk layer. This is because walking can always be performed in both directions.

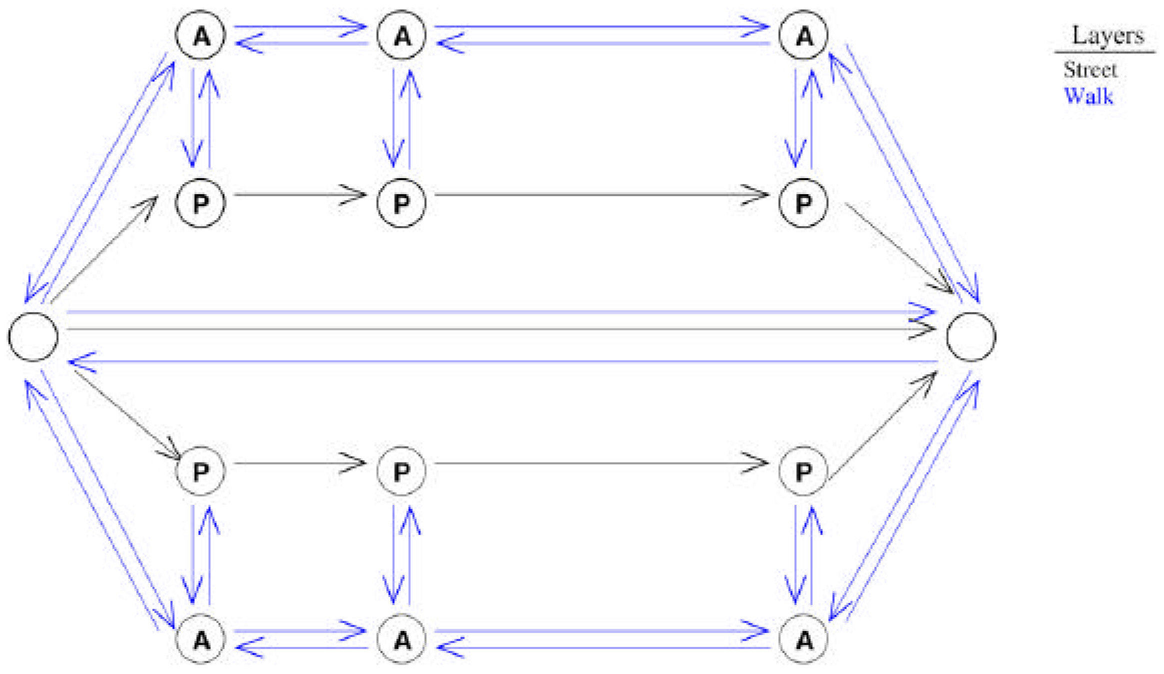

Fig. 8. This figure shows that there are only edges in one direction on the street layer.

3.4 Notes about the Network Assumptions made by the Route Planner

- No location (parking or activity) is located off the end of its link. Their locations on a link in TRANSIMS are specified by the distance from the endnode of the link (the value called "offset"). The Route Planner assumes that no offset is negative and that every offset is less than the length of the corresponding link. If this assumption is not satisfied, the Route Planner prints warnings. It proceeds in planning, but its behavior (especially with respect to calculating distances) is not defined.
- No two parking locations lie on the same link and have the same offsets.
- Each activity location is adjacent to a parking location. (This is not important if no trips are planned from the activity location that starts with wc or to the activity that ends with cw). This restriction will be removed in a future version of the Route Planner.
- No link has length 0.

3.5 Transit

Information about the transit system comes from the TRANSIMS Network transit stop table (configuration file key: `NET_TRANSIT_STOP_TABLE`), the transit route file (configuration file key: `TRANSIT_ROUTE_FILE`), and the transit schedule file (configuration file key: `TRANSIT_SCHEDULE_FILE`).

Each transit stop in the transit stop table is represented by a node in the transit layer for each type of transit that serves that stop. Each route in the route file has its own layer, containing a node for each stop on the route called route nodes. There are process links connecting each transit stop to the corresponding route nodes. The route nodes are connected by links, in the order that the route nodes appear in the route file.

Each transit stop must be explicitly connected to the walk network with process links to appropriate activity locations. The delays for the route links are taken from the route schedule file. The delays for these links are represented by a piecewise constant delay function.

Fig. 9 shows the TRANSIMS Network representation of two streets with bus stops and two bus routes connecting them.

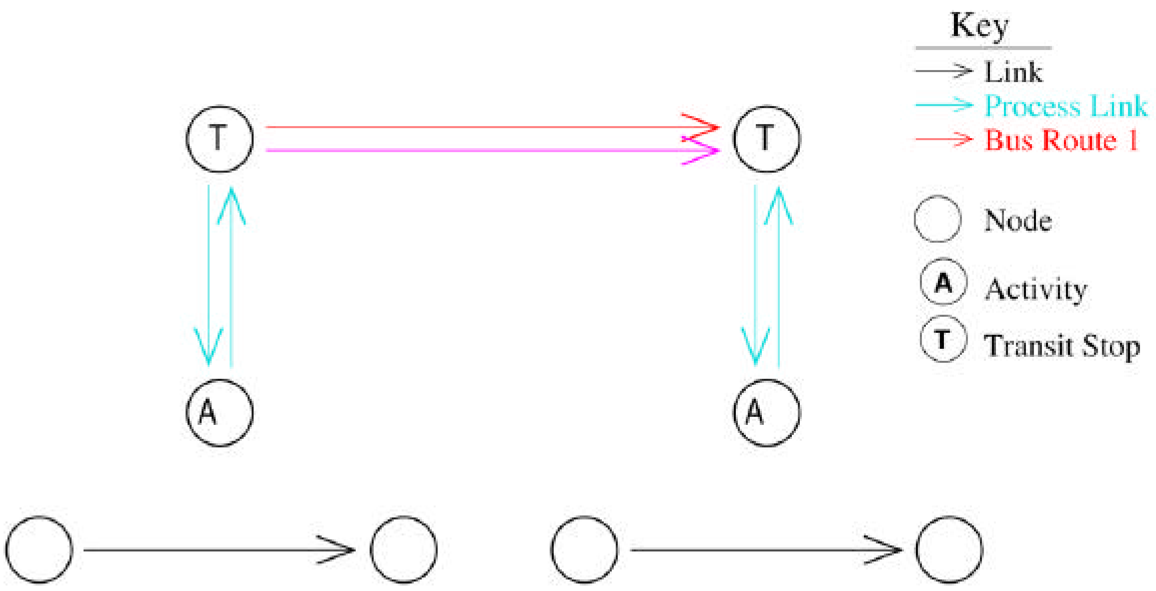

Fig. 9. TRANSIMS Network representation of two bus routes.

Fig. 10 shows the corresponding internal network representation. Note that there are five different layers in the internal network:

- the street layer containing the intersection nodes,
- the walk layer containing the activity locations,
- the bus layer containing the bus stops, and

- a layer for each bus route.

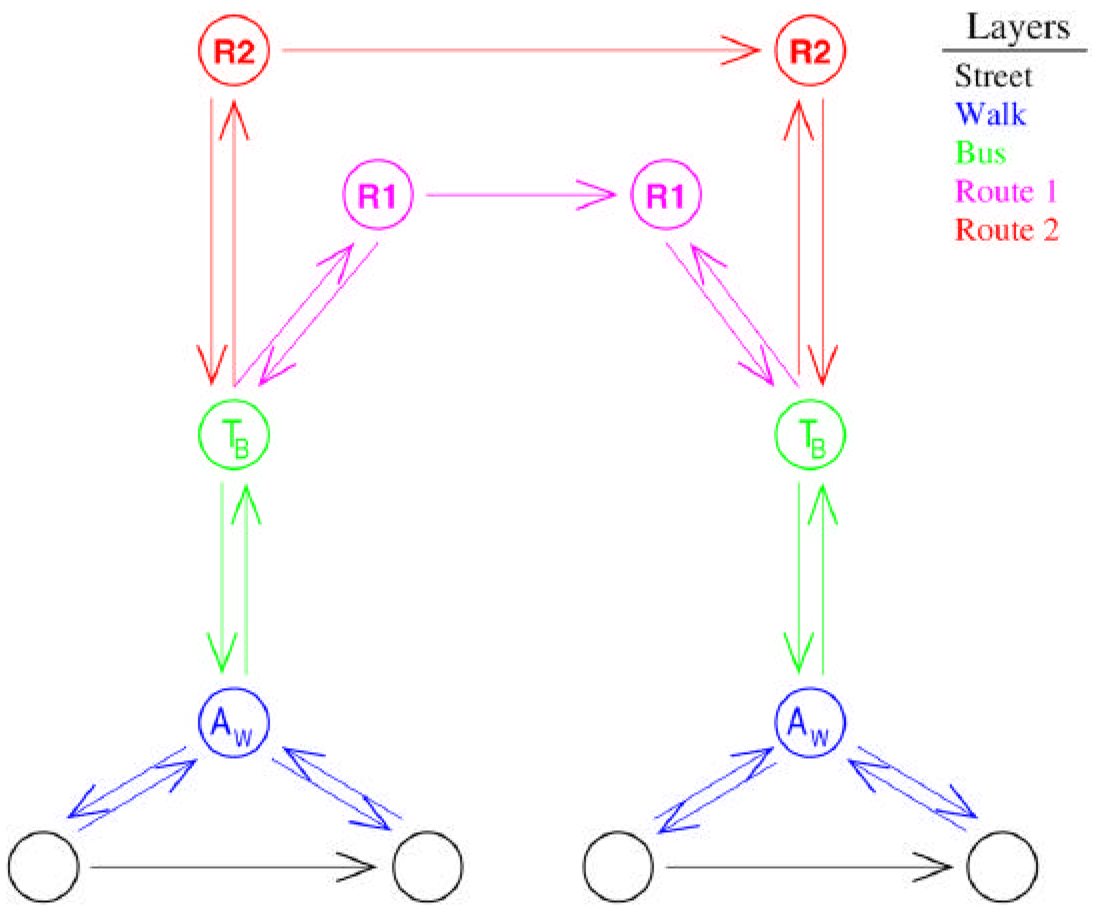

Fig. 10. Internal Route Planner network representation corresponding to Fig. 9.

Fig. 11 and Fig. 12 show a more complex example with two bus routes and a light rail line. Note that there is only one transit stop for both bus and light rail in the TRANSIMS Network, but separate stops for different types of transit in the internal network.

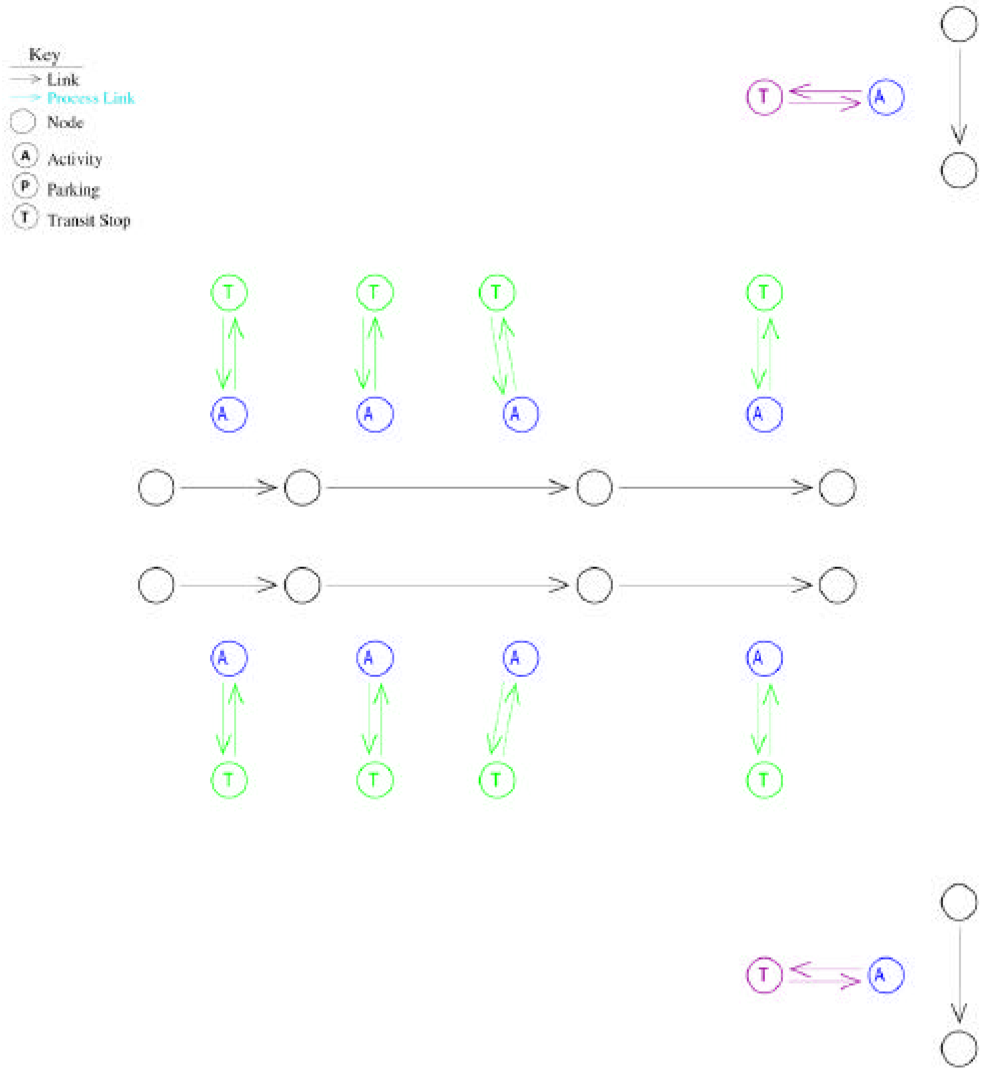

Fig. 11. TRANSIMS Network representation of a complex transit network with two bus routes and a light-rail line.

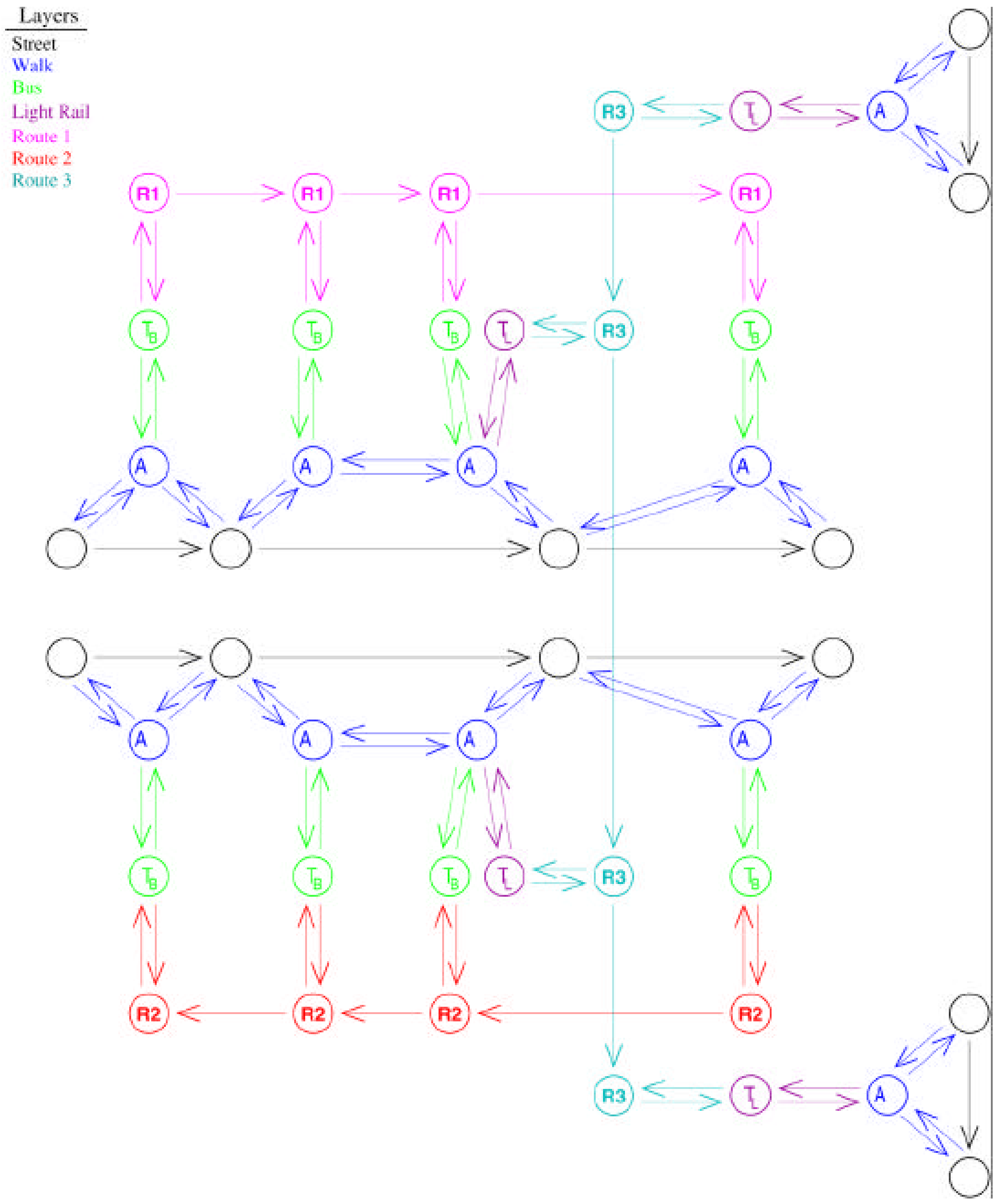

Fig. 12. Internal Route Planner network representation corresponding to Fig. 11.

3.5.1 Cost

Each link has a delay associated with it. Links on the street layer have a delay for driving on that link. Links on the walk layer have a delay for walking on that link. Transit links have a delay for the time between boarding a transit vehicle at one stop and exiting the vehicle at the following stop. Delays can either be constant, such as walking delays, or dependant on the time of day.

The default delay for a street link is the free speed delay. It is calculated from the free speed on that link and the length of the link. The actual delays calculated by the Traffic Microsimulator are used to provide more accurate information. These delays are given in the link delay file (configuration file key: `ROUTER_LINK_DELAY`). Each delay represents the average delay experienced for the vehicles that traversed the link, averaged over a 15-minute interval.

The delay for walking or biking on a link is determined from the length of the link and the walking speed (configuration file key: `ROUTER_WALKING_SPEED`) or biking speed (configuration file key: `ROUTER_BIKING_SPEED`). There are also delays for entering transit vehicles (configuration file key: `ROUTER_GET_ON_TRANSIT_DELAY`) and exiting transit vehicles (configuration file key: `ROUTER_GET_OFF_TRANSIT_DELAY`).

Process links can also have a delay associated with them. For example, the delay involved in parking a vehicle in a lot can be represented by the delay on the process link from the parking location to any adjacent activity locations.

To increase the effectiveness of microsimulation-Route Planner feedback, noise can be added to the link delays. The maximum amount of noise to add to the link as percentage of the link delay can be specified (configuration file key: `ROUTER_NOISE_DELAY`). If the delay for a link is d and the specified noise percentage value is n, the reported delay will be in the interval $(d\text{-}nd, d+nd)$.

3.5.1.1 Heuristics

To increase performance, the links that the Route Planner examines can be reduced. This is done by artificially increasing the delay for links that lead in the wrong direction. For example, assume that the source location for a trip is in the southern part of the network and that the destination location is directly north. Links that head north will be preferred over links that lead east or west. The farther from north that a link leads, the less likely it is that the link will be considered.

The *Sedgewick-Vitter heuristic* can be used for Euclidean graphs. The heuristic allows finding almost optimal shortest paths between nodes in a Euclidean graph. A parameter called `overdo` (configuration file key: `ROUTER_OVERDO`) allows for a tradeoff between the running time and optimality of the paths found. The internal network is not strictly Euclidean, since only certain nodes may be reached from each node (the graph is not complete), but we have found that the paths produced with moderate values, such as `overdo = 3`, look quite realistic and bring a considerable improvement to running time.

However, if this heuristic is used, the plans will be less sensitive to feedback. The larger the value of `overdo`, the longer congestion will be tolerated by the Route Planner before alternative routes are taken.

3.5.1.2 Monetary Cost

In addition to travel time delay, process links can also have an associated monetary cost. This can be used to account for parking fees and transit fares. Additionally, transit routes may have zone-to-zone fares associated with them. Currently, the Route Planner does not use monetary costs.

3.5.1.3 Generalized Cost Function

To more accurately model mode choice, the concept of a generalized cost function (GCF) has been developed. The GCF allows other factors, besides travel time and monetary cost, to be taken into account when determining a plan for a traveler. These other factors are included in the "cost" of a trip. The importance of the monetary cost of a trip may be modified depending on a traveler's income. A greater penalty for traveling on congested links can be imposed by calculating the difference between actual delay and free speed delay. Transit transfers may impose a higher cost than the actual delay involved. The GCF currently used is simply the travel time.

4. VEHICLE LIBRARY

4.1 Overview

This section gives the protocol for the interaction of the TRANSIMS vehicle library with the TRANSIMS Route Planner and Traffic Microsimulator. The TRANSIMS Population Synthesizer generates and assigns private vehicles to households. The Activity Generator assigns a set of possible vehicles to each member of a household.

Freight and transit vehicles (and the plans for their drivers) are generated by separate utilities, but these must be included in the vehicle database. The vehicle IDs assigned by these utilities must be unique.

4.1.1 File Format

Fields in the vehicle file are tab- or space-delimited. Each line of the vehicle file contains four mandatory fields:

- household ID,
- vehicle ID,
- ID of the starting location, and
- the TRANSIMS Network type of the vehicle.

The TRANSIMS Network vehicle type must be one of the following values:

1 =	Auto
2 =	Truck
4 =	Taxi
5 =	Bus
6 =	Trolley
7 =	StreetCar
8 =	LightRail
9 =	RapidRail
10 =	RegionalRail

The line may contain optional integer fields whose meaning is user defined. The number of these identifier fields may vary among different vehicle files. The number of optional identifier fields must be the same on every line within a vehicle file. The value *-1* is used as a default placeholder value for both the starting location and optional integer fields when the values are unknown or unused.

4.1.2 Format

```
<Household ID> <Vehicle ID> <starting location> <network type> [<int 1> ... <int n>]
```

4.1.3 Example

Household 1460 has two vehicles (500100 and 500101); both start at the home location (78) and are of network type auto (1). Two optional user-defined integer fields are present in this file.

The first field is the emissions vehicle type (10), which is the same for both vehicles. The second integer field is an indicator of the maintenance level of the vehicle. Note that the second vehicle (500101) has unknown/unused value (-1) for the second integer field.

```
   1460 500100 78 1 10 30
   1460 500101 78 1 10 -1
```

4.2 Vehicle Library Files

Appendix A contains vehicle library files and provides and example for review.

5. Plan Files

5.1 Overview

This section gives the protocol of the TRANSIMS plan file interface between the Route Planner and the Traffic Microsimulator.

5.2 File Format

The TRANSIMS code supplies a library of C routines, as well as a TPlan C++ object that can read and write this format.

The format consists of a required “header” and a set of “mode-dependent data.” The header contains information common to every kind of leg. Code that uses the plans may choose to ignore some or all of the mode-dependent data. For example, the Traffic Microsimulator will not simulate walking or bicycling, but it will use the estimated duration from the planner.

Because the origin, destination, and expected duration of any leg are available in the header information, the simulation does not require any data in the mode-dependent part of a walk leg.

5.2.1 Data Definitions and Format

A plan file contains a series of records, each of which specifies a single leg of a traveler’s trip. Each record contains the fields shown in the table found in Appendix B, in the order shown, separated by white space [space(s), tab(s), and/or a single newline]. The field names are not written in the data file. There is a blank line separating each pair of records. The file is written in ASCII text. Efficiency concerns are addressed by accessing plan files through an index. See the *Index* section for details.

The combination of Duration, Stop Time, and Max Time Flag allows flexible specification of departure times. For example, attending a movie might be encoded as follows:

```
duration = 0 seconds;
stop time = 20*3600 + 30*60 = 73800;
maxTime = true;
```

which means, “this activity ends at 8:30 P.M., or as soon as the traveler arrives, whichever is later.”

Similarly, *work* might be encoded as follows:

```
duration = 8 hours;
stop time = 17*3600 = 61200;
maxTime = true;
```

which means “stay at work until 5:00 P.M., or eight hours after arrival, whichever is later.”

Shopping at lunch might be encoded as follows:

```
duration = 0.5 hours;
stop time = 12*3600 + 45*60 = 45900;
maxTime = false;
```

which means “shop for half an hour or until 12:45 P.M., whichever is earlier.”

5.2.2 Mode-Dependent Data

Mode-dependent data are written by the Route Planner and interpreted by the Traffic Microsimulator. Appendix C provides such data for review.

5.3 Plan Library Files

Table 6 records plan library files.

Table 6. Plan library files.

Type	File Name	Description
Binary Files	*libTIO.a*	TRANSIMS Interfaces library.
Source Files	*planio.c*	Defines plan data structures and interface functions.
	planio.h	Plan interface functions source file.

5.4 Configuration File Keys

Appendix D records the plan file configuration file keys.

5.5 Example

Appendix E gives a six-leg plan for traveler 1. The plan consists of a walk-car-walk-bus-walk scenario.

Appendix A: Vehicle Library Files

Type	File Name	Description
Binary Files	*libTIO.a*	The TRANSIMS Interfaces library
Source Files	*vehio.h*	Defines vehicle data structures and interface functions
	vehio.c	Vehicle interface functions source file

Example

Read all of the data in the vehicle file, then write the vehicle information to another file. The data for each vehicle is stored in a `VehicleData` structure.

```
#include <stdio.h>
#include <vehio.h>

int main(int argc, char *argv[])
{
   FILE   *fp;
   FILE   *outfp;
   int    count = 0;
   const VehicleData      *veh;

   if (argc < 3) {
      fprintf(stdout, “Usage:  testveh <veh input file> <output file>\n”);
      exit(0);
   }

   fp = fopen(argv[1], “r”);
   if (fp == NULL) {
      printf(“Failed to open file %s...exiting\n”, argv[1]);
      exit(0);
   }

   outfp = fopen(argv[2], “w”);
   if (outfp == NULL) {
      printf(“Failed to open file %s...exiting\n”, argv[2]);
      exit(0);
   }

   while (moreVehicles(fp)) {
      veh = getNextVehicle(fp);
      if (veh == NULL) {
         fprintf(stderr, “Error FAILED to get vehicle...exiting\n”);
         break;
      }
      count++;
      if (!writeVehicle(outfp, veh)) {
         fprintf(stderr, “Failed to write vehicle %d\n”, veh>fVehicleId);
      }
   }

   fclose(fp);
   fclose (outfp);
   return 0;
}
```

Appendix B: Plan Data Definitions and Format

Column Name	Description	Allowed Values
Traveler (Person) ID	Each person is given a unique ID in the population file.	integer
User Field	Available to the user to set as desired. Its value is not used internally by the simulation, but is passed to the output system for use in filtering.	integer
Trip ID	Numbers the trips for the traveler sequentially from 1. The trip ID is not used by the simulation.	unsigned 16-bit integer
Leg ID	Numbers the legs within a trip sequentially from 1.	integer
Activation Time	The earliest time the simulation needs to worry about this leg. It is generally the starting time (estimated by the Route Planner) for a leg. For a transit leg, however, it represents the arrival time of the passenger at the transit stop, rather than the arrival time of the transit vehicle.	integer: seconds since midnight
Start Accessory ID	Denotes the network accessory ID of the starting location for this leg.	unsigned long
Start Accessory Type	Denotes the type of accessory of the corresponding location. It is necessary because the IDs are not globally unique over accessories. It should be one of: 1) `activity location` 2) `parking` 3) `transit stop` as defined in **`TNetAccessory::EType`** of *NET/Accessory.h*.	integer enumeration
End Accessory ID	As above, except it is for the destination rather than the starting accessory.	unsigned long, integer
End Accessory Type	As above, except it is for the destination rather than the starting accessory.	unsigned long, integer
Duration	In conjunction with Stop Time and Max Time Flag, specifies how long this leg is expected to take.	integer: seconds
Stop Time	In conjunction with Stop Time and Max Time Flag, specifies an absolute ending time for this leg.	integer: seconds since midnight
Cost	Monetary cost of the trip, in cents.	Integer
GCF	Generalized Cost Function. This is the value that the Route Planner attempts to minimize when planning travelers. Currently the same as duration.	Integer

Column Name	Description	Allowed Values
Max Time Flag	If true, the end of this activity is best estimated as *max(start time + duration; stop_time).* Otherwise, use the minimum instead. In the simulation, the actual start time is used, rather than the estimated activation time.	boolean
Driver Flag	True, if the traveler is driving a vehicle on this leg.	boolean
Mode	Mode of travel. This, together with the driver flag, determines the interpretation of the mode-dependent data following the header. Currently, it must be one of: 0 - `car` 1 - `transit` 2 - `pedestrian` 3 - `bicycle` 4 - `non-transportation activity` as defined in the **`TPlan::ETravelMode enum`** of *PLAN/Plan.h*.	integer, enumeration
Number of Tokens	Number of white-space-separated tokens in the mode-dependent data block (not including this field itself).	integer

Appendix C: Mode-dependent Data

Table 7. Mode-dependent data for a car driver.

Data	Description	Allowed Values
Vehicle ID	Each vehicle (with its ID) available in the simulation is listed in the vehicle database.	integer
Number of Passengers	The number of passengers, not including the driver, on this leg.	integer
List of Node IDs	The nodes (in order) through which the driver's route will pass.	integer
List of Passenger IDs	The traveler ID of each passenger to be carried on this leg.	integer

Table 8. Mode-dependent data for a car passenger.

Data	Description	Allowed Values
Vehicle ID	Each vehicle (with its ID) available in the simulation is listed in the vehicle database.	integer

Table 9. Mode-dependent data for a transit driver.

Data	Description	Allowed Values
Schedule Pairs	Number of (stop ID, depart time) pairs	Integer
Vehicle ID	Each vehicle (with its ID) available in the simulation is listed in the vehicle database.	Integer
Route ID	Route IDs are specified in the transit route file. Only one route ID is allowed per leg.	Integer
List of Node IDs	The nodes (in order) through which the driver's route will pass.	Integer
List of Schedule Pairs	Each pair consists of a stop ID and a depart time. When a transit driver arrives at a transit stop whose ID is given in this list, the driver will remain at that stop until the depart time.	Integer, integer

Table 10. Mode-dependent data for a transit passenger.

Data	Description	Allowed Values
Route ID	Traveler will board any transit vehicle whose driver's plan matches this Route ID.	integer

Table 11. Mode-dependent data for a pedestrian.

Data	Description	Allowed Values
List of Node IDs	The nodes (in order) through which the traveler's route will pass.	integer

For activity legs, there is no mode-dependent data.

Appendix D: Plan File Configuration File Keys

Configuration File Key	Description
`CA_USE_PARTITIONED_ROUTE_FILES`	If this key is set, the simulation expects to find separate indexes into a plan file for each slave. These can be produced using a partition file and the *DistributePlans* utility.
`PLAN_FILE`	Location of a file containing plans, or the base name of an index that points to plan files. Used by the Route Planner for output and the Traffic Microsimulator and Selector/Iteration Database for input.

Appendix E: Annotated Example of a Plan

Trip/Leg	Plan	Description

Trip 1/Leg 1	`1 156 1 1 1 0`
	`25200 123 1 456`
	`33 25200 1`
	`0 2`
	`2`
	`1000 1001`
		The user has chosen to mark this leg with the code 156, which has meaning only to that user but will be duly reported in any output concerned with this leg. It is trip 1, leg 1 for this traveler. It is the first leg to be simulated for this traveler, but not the last. The planner expects the trip to start at 25200 = 7∗3600 = 7 AM. The leg will start at activity location 123 and end at parking accessory 456. The planner expects the trip to take 33 seconds. The traveler's next leg will begin upon arrival at the destination or 33 seconds after departure from the origin, whichever is later. The traveler is not driving a vehicle and is, in fact, walking (mode = 2). There are two tokens of mode-dependent data, which in this case might be the nodes traversed. The CA microsimulation would probably simply use the planner's estimated duration and place the traveler in the destination queue 33 seconds after his arrival at the origin. However, the simulation could also choose to estimate its own duration. The microsimulation will not use the node information.

Trip 1/Leg 2	`1 156 1 2 0 0`
	`25233 456 2 789`
	`1314 0 1`
	`1 0`
	`18`
	`0 0`
	`1 2 3 4 5 6 7 8 9`
	`10 11 12 13 14 15 16`
		Leg 2 of trip 1 is neither the first nor the last. The traveler will be driving (driver flag = 1) a car (mode = 0) from parking accessory 456 to parking accessory 789 via the 16 nodes 1-16 using vehicle 0, carrying no passengers. The expected start time is 7:00:33 a.m., and the expected duration is 1314 seconds.

Trip 1/Leg 3	`1 156 1 3 0 0`
	`26547 789 2 10 2`
	`127 0 1`
	`1 0`
	`5`
	`0 1`
	`17 18`
	`1000`
		Traveler 1 picks up one passenger (traveler 1000) and drives to parking accessory 10 via nodes 17 and 18.

Trip 1/Leg 4	`1 156 1 4 0 0`
	`26674 10 2 11 3`
	`30 0 1`
	`0 2`
	`0`
		The traveler walks (mode = 2) from parking accessory 10 to bus stop (accessory type = 3) 11. The planner, knowing that the simulation will not simulate walking, has chosen not to write out the details of the path the walker will take (Number Of Tokens = 0).

Trip 1/Leg 5	`1 156 1 5 0 0`
	`26704 11 3 4 3`
	`1502 0 1`
	`0 1`
	`1`
	`72`
		The traveler will ride in (driver_flag = 0) the first bus (mode = 1) arriving on route 72, from bus stop 11 to bus stop 4.

Trip 1/Leg 6	`0 156 1 0 0`
	`28206 4 3 5 1`
	`31 0 1`
	`1 2`
	`0`
		The traveler takes 31 seconds to walk from bus stop 4 to activity location 5.

Trip 2/Leg 1	`1 156 2 0 1`
	`28237 5 1 5 1`
	`28800 61200 1`
	`1 4`
	`0`
		This is the first leg of trip 2 for traveler 1. Since the last leg flag is set, it is also the last leg that will be simulated. It is an activity (mode = 4) that ends at 5:00 p.m. (= 17 ∗ 3600 = 61200 seconds) or eight hours (= 8 ∗ 3600 = 28800) after arrival, whichever is later. There is no data associated with this leg, although the planner could, in principle, add anything—a list of projects the person will be working on, a list of groceries to buy, etc.

Chapter Four: Index